Dokumentacja - "Refleks"

Skład grupy (G02, czwartek 16:15):

Jakub Sikora 251626
Adrian Jagieła 251531
Patryk Krawczyk 251563

Systemy Wbudowane 2025

Spis treści

1. Opis płytki rozwojowej z mikrokontrolerem LPC1343

1.1. Charakterystyka ogólna

Płytka rozwojowa z mikrokontrolerem LPC1343 (np. LPC-P1343 lub LPC-H1343) to uniwersalne narzędzie przeznaczone do szybkiego prototypowania i testowania aplikacji wbudowanych. Mikrokontroler LPC1343 firmy NXP, oparty na 32-bitowym rdzeniu ARM Cortex-M3, zapewnia wysoką wydajność, niski pobór mocy oraz szeroki zestaw peryferiów, co czyni go idealnym wyborem dla projektów wymagających integracji wielu funkcji na jednej płytce.Posiada zegar kwarcowy 12 MHz podłączony do pinów XTALIN/XTALOUT.

1.2. Główne cechy mikrokontrolera LPC1343

- **Rdzeń:** ARM Cortex-M3, 32-bitowy RISC, taktowanie do 72 MHz.
- **Pamięć:** 32 kB Flash (programowa), 8 kB SRAM.
- **Zasilanie:** 2,0–3,6 V (typowo 3,3 V).
- **Obudowa:** LQFP48.
- **Wejścia/wyjścia:** 42 linie GPIO z konfigurowalnymi rezystorami.
- **Interfejsy komunikacyjne:**
  - USB 2.0 full-speed device z wbudowanym PHY.
  - UART z obsługą RS-485, FIFO i generacją ułamkowych prędkości transmisji.
  - SSP (SPI) z FIFO i obsługą wielu protokołów.
  - I2C z obsługą Fast-mode Plus (do 1 Mbit/s), trybem monitorowania i rozpoznawaniem wielu adresów.
- **Timer i liczniki:** 4 uniwersalne timery/liczniki (łącznie 4 wejścia capture, 13 wyjść match).
- **ADC:** 10-bitowy przetwornik analogowo-cyfrowy (do 8 kanałów).
- **Inne peryferia:**
  - Watchdog Timer (WDT).
  - Wbudowany bootloader obsługujący ISP i IAP.
  - NVIC (Nested Vectored Interrupt Controller).
  - Brownout detect z czterema progami oraz Power-On Reset.

1.3. Bezpieczeństwo i użytkowanie

- Płytka dostarczana jest w opakowaniu antystatycznym – należy unikać ładunków elektrostatycznych podczas pracy.
- Do programowania wymagany jest odpowiedni programator/debugger (np. ARM-JTAG-EW, ARM-USB-OCD, SWD).
- Zalecane środowiska programistyczne to IAR for ARM, Rowley CrossWorks lub dedykowane IDE Eclipse-based (np. LPCXpresso).
- Przed podłączeniem płytki do komputera przez USB należy upewnić się, że nie występują zwarcia na liniach zasilania.

2. Podział pracy w zespole wraz z listą wykorzystanych funkcjonalności

Tabela 1. Procentowy podział pracy członków zespołu.

Imię i nazwisko	Udział (%)
Jakub Sikora	33.3(3)
Adrian Jagieła	33.3(3)
Patryk Krawczyk	33.3(3)

Tabela 2. Lista wykorzystanych funkcjonalności i odpowiedzialnych za nie autorów.

Funkcjonalność	Autor
Timer	Patryk Krawczyk
Dźwięk	Jakub Sikora
Joystick (GPIO)	Jakub Sikora
Wyświetlacz OLED	Adrian Jagieła
Interfejs SPI	Jakub Sikora
Linijka diod	Adrian Jagieła
Interfejs I2C	Jakub Sikora
Moduł EEPROM	Patryk Krawczyk
Akcelerometr	Patryk Krawczyk
Czujnik oświetlenia	Adrian Jagieła

3. Opis działania programu - instrukcja użytkownika

Gra toczy się przez pięć rund oraz rozpoczyna się od animowanej sekwencji startowej z motywami sci-fi ("Heat up...", "Thrust OK", "Fire laser", ...). Następnie wyświetlany jest ekran powitalny z bieżącym rekordowym wynikiem uzyskanym w grze w trakcie uprzednich rozgrywek. Użytkownik nawiguje przez główne menu za pomocą joysticka, wybierając spośród opcji: "Start game", "Reset score", "High score", "Credits"lub "Exit".

Aktualny postęp gry można obserwować na pasku diod LED, gdzie każda dioda reprezentuje ukończoną rundę oraz na wyświetlaczu 7-segmentowym. W każdej rundzie użytkownik widzi na ekranie obrys koła z komunikatem "Wait..."wraz z dźwiękiem ostrzegawczym. Po pseudolosowym odstępie czasu (0,5-3,5 sekundy) koło zostaje wypełnione, sygnalizując rozpoczęcie pomiaru czasu reakcji. Kiedy użytkownik naciśnie joystick, uzyskany wynik w milisekundach jest wyświetlany na ekranie wraz z informacją o ewentualnym nowym rekordzie ("NEW RECORD!") i charakterystycznym dźwiękiem zwycięstwa.

System automatycznie dostosowuje motyw kolorystyczny (jasny/ciemny) na podstawie oświetlenia otoczenia. Opcja zresetowania zapisanego w pamięci rekordowego wyniku jest dostępna poprzez przechylenie płytki o kąt 30 stopni w dowolnym kierunku z poziomu ekranu głównego - operacja zostaje potwierdzona komunikatem "Reset HS"na wyświetlaczu. Uzyskanie czasu równego 0 ms również skutkuje zresetowaniem rekordu. Po zakończeniu wszystkich pięciu rund wyświetlane są statystyki: średni czas reakcji oraz najlepszy wynik z bieżącej sesji.

4. Opis działania programu - opis algorytmu

1. Inicjalizacja wszystkich potrzebnych urządzeń peryferyjnych oraz konfiguracja systemu audio z amplifikatorem LM4811.
2. Kalibracja akcelerometru w pozycji neutralnej i ustawienie początkowego motywu kolorystycznego.
3. Wyświetlenie animowanej sekwencji startowej z komunikatami sci-fi i efektami dźwiękowymi.
4. Prezentacja głównego menu z nawigacją za pomocą joysticka i obsługą następujących opcji:
   (a) **Start game**: Rozpoczęcie 5-rundowej rozgrywki
   (b) **Reset score**: Resetowanie rekordu do wartości domyślnej (9999 ms)
   (c) **High score**: Wyświetlenie obecnego rekordu
   (d) **Credits**: Wyświetlenie informacji o autorze z melodią.
   (e) **Exit**: Zakończenie programu z sekwencją dźwięków pożegnalnych
5. Rozpoczęcie 5-iteracyjnej pętli gry:
   (a) Ustawienie odpowiedniej diody LED wskazującej numer rundy (0-4).
   (b) Dynamiczne dostosowanie motywu kolorystycznego na podstawie czujnika światła.
   (c) Wyświetlenie obrysu koła z komunikatem "Czekaj..."i odtworzenie dźwięku ostrzegawczego.
   (d) Generacja pseudolosowego opóźnienia (500-3500 ms) na podstawie ziarna z czujnika światła.
   (e) Wypełnienie koła jako sygnał wizualny i rozpoczęcie precyzyjnego pomiaru czasu reakcji za pomocą Timer32.
   (f) Oczekiwanie na naciśnięcie joysticka z i walidacją czasu reakcji.
   (g) Wyświetlenie zmierzonego czasu reakcji z jednostką "ms".
   (h) Porównanie z rekordem przechowywanym w EEPROM i aktualizacja w przypadku poprawy wyniku.
   (i) Wyświetlenie komunikatu "NEW RECORD!"z dźwiękiem zwycięstwa dla nowych rekordów.
   (j) Oczekiwanie na potwierdzenie kontynuacji przez użytkownika.
6. Wyświetlenie końcowych statystyk gry:
   (a) Komunikat "Game Complete!"
   (b) Średni czas reakcji z wszystkich pięciu rund
   (c) Aktualny najlepszy wynik (rekord)
7. Powrót do głównego menu dla kolejnej rozgrywki.
8. Ciągłe monitorowanie przechylenia płytki za pomocą skalibrowanego akcelerometru - reset rekordu przy przekroczeniu progu ± 30 jednostek na osiach X lub Y.

5. Opis funkcjonalności poszczególnych elementów

5.1. Timer

Na płytce znajdują się dwa 32-bitowe czasomierze (ozn. timer0 i timer1). Oba z nich zostały wykorzystane w programie. Czasomierze działają na zasadzie zliczania cykli zegara peryferyjnego (PCLK), zapewnianego przez zegar systemowy.

W programie wykorzystane są następujące rejestry timerów:

5.1.1. TCR (Timer Control Register)

- Adres: `0x004`.
- 32-bitowy rejestr kontrolujący stan rejestrów TC i PC.
- Zawiera bity sterujące służące do:
  - Uruchamiania/zatrzymywania rejestrów TC i PC (bit 0 – Counter Enable). Kiedy wartość bitu jest równa 1, rejestry są aktywne. W przeciwnym wypadku, zostają zdezaktywowane.
  - Resetowania rejestrów TC oraz PC (bit 1 – Counter Reset). Kiedy wartość bitu jest równa 1, w kolejnym cyklu zegara PCLK następuje reset rejestrów. Rejestry pozostają zresetowane do czasu ustawienia wartości bitu na 0.

5.1.2. TC (Timer Counter)

- Adres: `0x008`
- 32-bitowy rejestr inkrementowany co PR+1 cykli zegara peryferyjnego (PCLK).
- Jest kontrolowany przez rejestr TCR.
- Wartość w TC odzwierciedla liczbę przepełnień rejestru PC timera od momentu jego uruchomienia.

5.1.3. PR (Prescale Register)

- Adres: `0x00C`.
- 32-bitowy rejestr definiujący maksymalną wartość rejestru PC.
- Kiedy wartość rejestru PC osiągnie wartość równą tej w rejestrze PR, następuje inkrementacja rejestru TC i wyczyszczenie rejestru PC.

Uwaga: Zegar systemowy pracuje z częstotliwością 72 MHz. Poniżej znajduje się fragment kodu programu ustawiający tę wartość:

```
#define __XTAL (12000000UL) /* Oscillator frequency */
#define __SYS_OSC_CLK (__XTAL) /* Main oscillator frequency */
#define SYSPLICTRL_Val 0x00000025
LPC_SYSCON->SYSPLLCTRL = SYSPLLCTRL_Val;
SystemCoreClock = __SYS_OSC_CLK * ((LPC_SYSCON->SYSPLLCTRL & 0x01F) + 1);
/* 12MHz * (5+1) = 72MHz */
```

$$12\,\text{MHz} \times (0x25\&0x1F + 1) = 12\,\text{MHz} \times (5 + 1) = 72\,\text{MHz}$$

5.1.4. PC (Prescale Counter)

- Adres: `0x010`.
- 32-bitowy rejestr zliczający.
- Wartość rejestru jest inkrementowana z każdym cyklem zegara PCLK.
- Kiedy wartość rejestru PC osiągnie wartość równą tej w rejestrze PR, w kolejnym cyklu zegara PCLK następuje inkrementacja rejestru TC i wyczyszczenie rejestru PC.

5.1.5. MCR (Match Control Register)

- Adres: `0x014`.
- 32-bitowy rejestr sterujący działaniem rejestrów MR.
- Umożliwia ustalenie operacji wykonywanej, gdy nastąpi dopasowanie rejestrów TC i rejestrów MR0-3.
- Bity sterujące 0-11 służą do doboru operacji podczas dopasowania rejestru TC z odpowiednim rejestrem MR.

5.1.6. IR (Interrupt Register)

- Adres: `0x000`.
- Służy do identyfikacji źródeł wygenerowanych przerwań.

5.1.7. Opis użycia w programie

Rozpoczęcie odliczania na czasomierzu i odczytanie wartości bieżącego upływu czasu:

```
init_timer32(0, 72000);
LPC_TMR32B0->TCR = 0x02;
uint32_t prescalerValue = ((SystemFrequency/LPC_SYSCON->SYSAHBCLKDIV) / 1000) - 1;
LPC_TMR32B0->PR = prescalerValue;
LPC_TMR32B0->MCR = 0x00;
LPC_TMR32B0->TCR = 0x01;
(...)
uint32_t reactionTimeMs = LPC_TMR32B0->TC;
```

Opis wartości w rejestrach:

- Wywołanie funkcji `init_timer32(0, 72000)`: inicjalizuje timer **timer32_0** z wartością dopasowania 72000, co przy częstotliwości zegara systemowego 72 MHz odpowiada dokładnie 1 ms (72000 taktów / 72 MHz = 0,001 s).
- Wartość w rejestrze TCR: 0x02 – reset i zatrzymanie liczników rejestrów TC oraz PC.
- Wartość w rejestrze PR: częstotliwość zegara systemowego (SystemFrequency/SystemCoreClock) podzielona przez 1000 (by uzyskiwać wynik w milisekundach), a następnie dekrementowana o 1 (ponieważ rejestr TC jest inkrementowany co PR+1 cykli PCLK).
- Wartość w rejestrze MCR: 0x00 – reset rejestru / brak czynności przy dopasowaniu rejestrów MR i rejestru TC.
- Wartość w rejestrze TCR: 0x01 – uruchomienie liczników rejestrów TC oraz PC.

Funkcja `init_timer32(uint8_t timer_num, uint32_t timerInterval)` konfiguruje 32-bitowy timer mikrokontrolera LPC13xx (lub podobnego) poprzez ustawienie odpowiednich rejestrów sterujących.

- `timer_num` – wybór timera (0 dla `TMR32B0`, 1 dla `TMR32B1`),
- `timerInterval` – wartość dopasowania (Match), np. `72000` dla 1 ms przy zegarze 72 MHz.

Rejestr	Wartość	Opis
TCR	0x02 → 0x00 → 0x01	Reset licznika (TC, PC), a następnie uruchomienie timera
PR	(SystemCoreClock / 1000) - 1	Ustawienie preskalera – tak, aby 1 zliczenie TC odpowiadało 1 ms
MR0	timerInterval	Rejestr dopasowania – liczba zliczeń przed akcją (np. 72000 dla 1 sekundy)
MCR	0x03	Bit 0: przerwanie Bit 1: reset licznika przy dopasowaniu
IR	0xFF	Wyczyszczenie flag przerwań
NVIC_EnableIRQ	–	Aktywacja przerwań timera w kontrolerze NVIC

Tabela 5.1: Rejestry konfigurowane przez `init_timer32()`

Opis wykorzystywanej w programie funkcji `delay32Ms()`:

```
1     LPC_TMR32B0->TCR = 0x02; /* reset timer */
2     LPC_TMR32B0->PR = 0x00; /* set prescaler to zero */
3     LPC_TMR32B0->MR0 = delayInMs * ((SystemFrequency/LPC_SYSCON->SYSAHBCLKDIV) / 1000);
4     LPC_TMR32B0->IR = 0xff; /* reset all interrupts */
5     LPC_TMR32B0->MCR = 0x04; /* stop timer on match */
6     LPC_TMR32B0->TCR = 0x01; /* start timer */
7     /* wait until delay time has elapsed */
8     while (LPC_TMR32B0->TCR & 0x01);
```

Gdzie `delayInMs` = żądany czas oczekiwania w milisekundach.
Opis wartości w rejestrach:

- Wartość w rejestrze IR: 0xff – zresetowanie wszystkich przerwań, zgłoszonych przez rejestry MR0-3.
- Wartość w rejestrze PR: 0x00 – inkrementacja licznika rejestru TC w każdym cyklu PCLK.

- Wartość w rejestrze MR0: żądany czas oczekiwania w milisekundach wymnożony przez częstotliwość zegara podzieloną przez 1000 (aby uzyskać wynik w milisekundach). Wartość w rejestrze SYSAHBCLKDIV jest równa 1.
- W rejestrze MCR bit 2 jest ustawiany na wartość 1 w celu zatrzymania liczników rejestrów TC oraz PC poprzez ustawienie wartości bitu 0 rejestru TCR na 0 przy dopasowaniu rejestrów TC oraz MR0.

Uwaga: w programie jest używana również funkcja `delay32Us()` działająca w taki sam sposób, z wyjątkiem dzielenia częstotliwości zegara przez 1000000 zamiast 1000, aby uzyskać wynik w mikrosekundach.

5.2. Dźwięk

5.2.1. Architektura systemu audio

Sygnał wejściowy jest przekazywany poprzez pin 2 na porcie GPIO1 (P1.2). Sygnał jest w formie sygnału PWM, co wymaga, aby w celu jego wytworzenia, naprzemiennie ustawiać pin P1.2 w stan wysoki i niski, tworząc w ten sposób falę prostokątną.
System audio składa się z dwóch głównych komponentów:

1. **Filtr dolnoprzepustowy PWM** – konwersja sygnału cyfrowego na analogowy
2. **Wzmacniacz analogowy LM4811** – wzmocnienie sygnału do poziomu wymaganego przez głośnik

5.2.2. Filtr dolnoprzepustowy PWM – analiza techniczna

Zasada działania

Filtr dolnoprzepustowy PWM realizuje funkcję **cyfrowo-analogowego przetwornika (DAC)** poprzez:

- Generację sygnału PWM o zmiennym wypełnieniu
- Filtrację dolnoprzepustową eliminującą składowe wysokoczęstotliwościowe
- Uzyskanie sygnału analogowego proporcjonalnego do wypełnienia PWM

Konfiguracja sprzętowa

Sygnał wejściowy jest przekazywany poprzez pin 2 na porcie GPIO1 (P1.2). Sygnał jest w formie sygnału PWM, co wymaga, aby w celu jego wytworzenia, naprzemiennie ustawiać pin P1.2 w stan wysoki i niski, tworząc w ten sposób falę prostokątną.

Inicjalizacja pinu wejściowego filtra i ustawienie trybu PIO1_2:

```
1 GPIOSetDir( PORT1, 2, 1 );  // Ustawienie P1.2 jako wyjscie
2 LPC_IOCON->JTAG_nTRST_PIO1_2 = (LPC_IOCON->JTAG_nTRST_PIO1_2 & ~0x7) | 0x01;
```

- Pin **P1.2** jest konfigurowany jako GPIO (funkcja 0x01)
- Domyślnie pin ten pełni funkcję JTAG_nTRST, dlatego wymagana jest rekonfiguracja

Generacja sygnału PWM

Pin jest ustawiany w stan wysoki i niski za pomocą następujących makr:

```
#define P1_2_HIGH() (LPC_GPIO1->DATA |= ((uint16_t)0x1<<2))
#define P1_2_LOW() (LPC_GPIO1->DATA &= ~((uint16_t)0x1<<2))
```

Algorytm generacji tonu:

```
P1_2_HIGH();
delay32Us(0, note / (uint32_t)2);  // Polowa okresu - stan wysoki
P1_2_LOW();
delay32Us(0, note / (uint32_t)2);  // Polowa okresu - stan niski
```

Gdzie `note` to żądany okres drgań dźwięku.

Parametry techniczne

- Częstotliwość tonu: $f = \frac{1}{\text{note}}$ [Hz]
- Wypełnienie PWM: 50% (symetryczna fala prostokątna)
- Rozdzielczość czasowa: 32 μs (funkcja delay32Us)
- Zakres częstotliwości: ograniczony przez rozdzielczość timera
- Wejściowy sygnał PWM jest konwertowany na wyjściowy sygnał analogowy.
- Sygnał wyjściowy jest przekazywany do wzmacniacza analogowego LM4811.

5.2.3. Wzmacniacz analogowy LM4811 – specyfikacja

Konfiguracja pinów sterujących

Inicjalizacja pinów wzmacniacza:

```
GPIOSetDir( PORT3, 0, 1 );   // P3.0 - CLK (zegar sterujący)
GPIOSetDir( PORT3, 1, 1 );   // P3.1 - UP/DN (kierunek zmiany głośności)
GPIOSetDir( PORT3, 2, 1 );   // P3.2 - SHUTDOWN (wyłączenie)

GPIOSetValue( PORT3, 0, 0 ); // CLK = 0 (stan spoczynku)
GPIOSetValue( PORT3, 1, 0 ); // UP/DN = 0 (zmniejszanie głośności)
GPIOSetValue( PORT3, 2, 0 ); // SHUTDOWN = 0 (wzmacniacz wyłączony)
```

Funkcje pinów sterujących

- **CLK (P3.0):** Sygnał zegarowy do sterowania głośnością (zbocze narastające aktywne)
- **UP/DN (P3.1):** Kierunek zmiany głośności (1=zwiększ, 0=zmniejsz)
- **SHUTDOWN (P3.2):** Kontrola zasilania (1=aktywny, 0=wyłączony)
- Wejściowy sygnał analogowy jest wzmacniany i przekazywany do głośnika, który emituje falę dźwiękową.

5.2.4. Ścieżka sygnału audio

1. **Generacja PWM** → Pin P1.2 generuje sygnał prostokątny o częstotliwości tonu
2. **Filtracja RC** → Filtr dolnoprzepustowy (R+C) wygładza sygnał PWM
3. **Wzmocnienie** → LM4811 wzmacnia sygnał analogowy
4. **Reprodukcja** → Głośnik konwertuje sygnał elektryczny na akustyczny

5.3. Joystick GPIO

Joystick jest urządzeniem peryferyjnym sterowanym za pomocą GPIO (General Purpose Input/Output). Pozycja joysticka jest odczytywana poprzez sprawdzenie stanu pinów GPIO, do których podłączone są przyciski i osie joysticka. Mikrokontroler skanuje piny i wykrywa zmiany stanu, które oznaczają ruch joysticka.

5.3.1. Konfiguracja pinów GPIO

Joystick znajduje się na porcie drugim mikrokontrolera i ma 4 piny, które w funkcji `joystick_init` mają wartość ustawioną na 0, co oznacza, że ich kierunek jest ustawiony na "wejście"(jeśli byłoby 1, oznaczałoby kierunek "wyjście"):

```
GPIOSetDir( PORT2, 0, 0 );
GPIOSetDir( PORT2, 1, 0 );
GPIOSetDir( PORT2, 2, 0 );
GPIOSetDir( PORT2, 3, 0 );
```

5.3.2. Odczyt stanu joysticka

Joystick korzysta z flag bitowych, które pozwalają na reprezentowanie wielu kierunków jednocześnie za pomocą operacji bitowych OR. Na przykład, jeśli zarówno kierunek góra, jak i prawo są naciśnięte, wartość łączna będzie `JOYSTICK_UP | JOYSTICK_RIGHT`, co daje `0x02 | 0x10 = 0x12`. Odczyt wartości flag bitowych odbywa się poprzez odczyt wartości na porcie GPIO:

```
    uint8_t status = 0;
    if (!GPIOGetValue( PORT2, 0)) {
        status |= JOYSTICK_CENTER;
    }
```

Podsumowanie

Dzięki takiemu podejściu możliwe jest rozpoznanie wielu stanów joysticka oraz ich kombinacji, co pozwala na intuicyjną obsługę urządzenia w programie.

5.4. Interfejs SPI

SPI (Serial Peripheral Interface) to szybki, synchroniczny interfejs szeregowy wykorzystywany do komunikacji pomiędzy mikrokontrolerami a różnego rodzaju układami peryferyjnymi, takimi jak wyświetlacze OLED, czujniki czy pamięci. Architektura SPI oparta jest na modelu master-slave, gdzie jeden układ (master) steruje transmisją, a pozostałe (slave) odpowiadają na żądania.

5.4.1. Podstawowe linie SPI

Typowa magistrala SPI składa się z czterech linii:

- **SCK (Serial Clock)** – linia zegara generowana przez mastera, synchronizująca transmisję.
- **MOSI (Master Out, Slave In)** – linia danych od mastera do slave'a.
- **MISO (Master In, Slave Out)** – linia danych od slave'a do mastera.
- **SS/CS (Slave Select/Chip Select)** – linia wyboru układu slave, aktywna w stanie niskim.

5.4.2. Zasada działania

Transmisja SPI jest pełnodupleksowa – dane mogą być przesyłane w obu kierunkach jednocześnie. Master inicjuje komunikację, generując sygnał zegarowy i aktywując wybranego slave'a przez obniżenie linii SS. W każdym cyklu zegara jeden bit jest przesyłany z mastera do slave'a (MOSI) i jednocześnie jeden bit z slave'a do mastera (MISO).

Tryby pracy (CPOL/CPHA)

SPI posiada cztery tryby pracy, określane przez polaryzację (CPOL) i fazę (CPHA) sygnału zegara:

- **Mode 0:** CPOL=0, CPHA=0 – dane próbkowane na narastającym zboczu zegara.
- **Mode 1:** CPOL=0, CPHA=1 – dane próbkowane na opadającym zboczu zegara.
- **Mode 2:** CPOL=1, CPHA=1 – dane próbkowane na opadającym zboczu zegara, zegar w stanie spoczynku wysoki.
- **Mode 3:** CPOL=1, CPHA=0 – dane próbkowane na narastającym zboczu zegara, zegar w stanie spoczynku wysoki.

Wybór odpowiedniego trybu zależy od wymagań urządzenia peryferyjnego. Niezgodność trybów po obu stronach uniemożliwi poprawną komunikację.

5.4.3. Konfiguracja SPI w LPC1343

W programie inicjalizacja interfejsu SPI wygląda następująco:

```
1 LPC_SYSCON->PRESETCTRL |= (0x1<<0);     // Wlaczenie zasilania peryferii SPI
2 LPC_SYSCON->SYSAHBCLKCTRL |= (1<<11);   // Taktowanie peryferii SPI
3 LPC_SYSCON->SSPCLKDIV = 0x02;           // Ustawienie dzielnika zegara SPI
4
5 LPC_IOCON->PIO0_8 &= ~0x07;
6 LPC_IOCON->PIO0_8 |= 0x01;  // SSP MISO
7 LPC_IOCON->PIO0_9 &= ~0x07;
8 LPC_IOCON->PIO0_9 |= 0x01;  // SSP MOSI
```

Pierwsze trzy linie włączają zasilanie i taktowanie oraz ustawiają dzielnik zegara dla peryferium SPI. Kolejne linie konfigurowane są piny mikrokontrolera do funkcji MISO i MOSI.

Uwaga praktyczna: W przypadku kilku urządzeń slave, każde z nich powinno mieć osobną linię SS, sterowaną przez mastera. W stanie nieaktywnym linie SS należy podciągnąć do stanu wysokiego za pomocą rezystorów, aby uniknąć przypadkowej aktywacji urządzeń i konfliktów na magistrali.

5.4.4. Topologie połączeń

- **Jeden master, jeden slave:** najprostsza konfiguracja, wszystkie linie są połączone bezpośrednio.
- **Jeden master, wiele slave:** master wybiera aktywnego slave'a przez odpowiednią linię SS. Pozostałe urządzenia pozostają nieaktywne.
- **Daisy-chain:** slave'y połączone szeregowo, dane przesyłane przez kolejne urządzenia, jeden wspólny sygnał SS.

5.4.5. Zalety i ograniczenia SPI

Zalety:

- Bardzo wysoka szybkość transmisji (do kilkudziesięciu Mb/s).
- Prosta implementacja sprzętowa i programowa.
- Pełny dupleks – jednoczesna transmisja i odbiór danych.

Ograniczenia:

- Brak standaryzacji protokołu warstwy wyższej (każde urządzenie może mieć własny sposób interpretacji danych).
- Konieczność osobnej linii SS dla każdego slave'a w klasycznej topologii.
- Brak możliwości adresowania urządzeń na magistrali (w przeciwieństwie do I2C).

5.4.6. Zastosowanie w projekcie

W projekcie interfejs SPI został wykorzystany do komunikacji z wyświetlaczem OLED. Dzięki wysokiej przepustowości możliwa jest szybka aktualizacja zawartości ekranu, co jest kluczowe w grach wymagających dynamicznego odświeżania grafiki.

Przykład inicjalizacji pinów dla SPI:

```
1 GPIOSetDir(PORT0, 0, 1 );
2 GPIOSetDir(PORT1, 10, 1 );
3 GPIOSetDir(PORT2, 7, 1 );
4 GPIOSetDir(PORT0, 2, 1 );
```

Powyższy kod ustawia odpowiednie piny jako wyjścia, co pozwala na poprawne sterowanie wyświetlaczem OLED przez magistralę SPI.

Podsumowanie: SPI jest jednym z najczęściej wykorzystywanych interfejsów do komunikacji z szybkimi peryferiami. Warto pamiętać o poprawnej konfiguracji linii SS oraz zgodności trybów pracy, aby zapewnić bezproblemową transmisję danych.

5.5. Wyświetlacz OLED

Wyświetlacz OLED użyty w projekcie ma rozdzielczość 96 x 64 piksele i pracuje w oparciu o interfejs SPI. Dzięki technologii OLED zapewnia głęboki kontrast oraz wyraziste kolory, umożliwiając prezentację czytelnych informacji na niewielkiej powierzchni ekranu. W projekcie zastosowano funkcję `oled_putChar`, która zamienia podany znak na odpowiednie piksele wyświetlane na ekranie, korzystając z tablicy `font5x7` zawierającej bitmapy znaków.

5.5.1. Budowa i podłączenie wyświetlacza

Wyświetlacz OLED o przekątnej 0,95"i rozdzielczości 96x64 piksele korzysta z palety barw RGB i jest wyposażony w kontroler SSD1331. Moduł komunikuje się poprzez interfejs SPI, a napięcie zasilania wynosi od 3,3 do 5 V. Biblioteka zapewnia funkcję oled_putChar, która zamienia podany znak na odpowiednie piksele, które są później wyświetlane na ekranie. Dzieje się to przy pomocy tablicy font5x7, która zawiera każdy znak char zapisany w poniższy sposób:

```
    {
    _XXX____,
    X___X___,
    X_______,
    _XXX____,
    ____X___,
    X___X___,
    _XXX____,
    ________
    }
```

5.5.2. Główne komponenty wyświetlacza OLED

- **Interfejs SPI** – umożliwia szybką komunikację z mikrokontrolerem oraz transfer komend i danych obrazu.
- **Kontroler SSD1331** – odpowiada za interpretację komend, zarządzanie pamięcią RAM wyświetlacza oraz generowanie sygnałów sterujących pikselami.
- **Pamięć RAM** – przechowuje dane obrazu, które mają być wyświetlane. Kontroler cyklicznie odczytuje te dane podczas odświeżania ekranu (typowa częstotliwość odświeżania to ok. 140 Hz).
- **Układ sterujący** – steruje jasnością i kolorem diod OLED na podstawie danych z RAM.
- **Panel OLED** – matryca diod generujących obraz.

5.5.3. Przepływ danych i obsługa wyświetlacza

1. **Przesyłanie komend** – mikrokontroler wysyła przez SPI komendy sterujące (np. ustawienia parametrów wyświetlania, reset, inicjalizacja).
2. **Transfer danych obrazu** – mikrokontroler przesyła dane obrazu (bitmapy, znaki) do pamięci RAM wyświetlacza.
3. **Cykl odświeżania** – kontroler SSD1331 pobiera dane z RAM i steruje diodami OLED, generując widoczny obraz.

5.5.4. Przykład kodu i inicjalizacji pinów

W kodzie projektu wyświetlacz obsługiwany jest przez funkcje, które zamieniają znaki na bitmapy i przesyłają je przez SPI:

```
/* Przyklad fragmentu fontu */
const uint8_t font5x7[][5] = {
    /* '?' */
    { 0x22, 0x01, 0x51, 0x09, 0x06 },
    /* kolejne znaki... */
};
```

Inicjalizacja pinów do obsługi wyświetlacza przez SPI:

```
GPIOSetDir(PORT0, 0, 1 );
GPIOSetDir(PORT1, 10, 1 );
GPIOSetDir(PORT2, 7, 1 );
GPIOSetDir(PORT0, 2, 1 );
```

Ustawienie pinu zasilania na niski stan (wyłączenie wyświetlacza):

```
GPIOSetValue(PORT1, 10, 0 );
```

Po inicjalizacji parametrów, włączenie wyświetlacza:

```
GPIOSetValue(PORT1, 10, 1 );
```

5.5.5. Wyświetlanie piksela

```
    /* page address */
         if(y < 8)  page = 0xB0;
    else if(y < 16) page = 0xB1;
    else if(y < 24) page = 0xB2;
    else if(y < 32) page = 0xB3;
    else if(y < 40) page = 0xB4;
    else if(y < 48) page = 0xB5;
    else if(y < 56) page = 0xB6;
    else            page = 0xB7;
    add = x + X_OFFSET;
    lAddr = 0x0F & add;             // Low address
    hAddr = 0x10 | (add >> 4);      // High address
    add = y>>3;                     // Divide by 8
    add <<= 3;                      // Multiply by 8
    add = y - add;                  // Calculate bit position
    mask = 1 << add;                // Left shift 1 by bit position
    setAddress(page, lAddr, hAddr); // Set the address (sets the page,
                                    // lower and higher column address pointers)
    shadowPos = (page-0xB0)*OLED_DISPLAY_WIDTH+x;
    if(color > 0)
        shadowFB[shadowPos] |= mask;
    else
        shadowFB[shadowPos] &= ~mask;
    writeData(shadowFB[shadowPos]);
```

1. **Wyznaczenie strony (page):** Wyświetlacze OLED używają tzw. adresowania stronicowego, które dzieli całą matrycę pikseli na strony (pages). Każda strona to zestaw 8 poziomych wierszy (pikseli) w pionie, ale obejmujący pełną szerokość wyświetlacza w poziomie (np. 64 kolumny). Na podstawie wartości y dobierany jest odpowiedni adres strony $0xB0$ do $0xB7$.
2. **Adres kolumny:**
   - `add = x + X_OFFSET`
   - `lAddr = add & 0x0F` – niższe 4 bity (adres kolumny niski)
   - `hAddr = 0x10 | (add » 4)` – wyższe 4 bity (adres kolumny wysoki)
3. **Pozycja bitu w bajcie:**
   - Obliczenie numeru bitu w bajcie strony: $bit = y \bmod 8$

- Utworzenie maski bitowej: `mask = 1 « bit`

4. **Ustawienie adresu w kontrolerze OLED:**
   - Wywołanie `setAddress(page, lAddr, hAddr)` ustawia wskaźnik pamięci OLED.
5. **Modyfikacja bufora cieni (`shadowFB`):**
   - Obliczenie pozycji w buforze:

$$shadowPos = (page - 0xB0) \cdot OLED\_DISPLAY\_WIDTH + x$$

     gdzie
     - `0xB0`: adres pierwszej strony
     - `OLED_DISPLAY_WIDTH`: ilość pikseli w poziomie
   - Jeśli `color > 0`, ustawienie bitu: `shadowFB[shadowPos] |= mask`
   - Jeśli `color == 0`, wyczyszczenie bitu: `shadowFB[shadowPos] &= ~mask`
6. **Zapis do wyświetlacza:** Funkcja `writeData()` wysyła zmodyfikowany bajt do kontrolera OLED, modyfikując tym samym wartość znajdującą się pod ustawionym wcześniej adresem strony w jakiej znajduje się piksel.

```
1   #define OLED_DATA()   GPIOSetValue( PORT2, 7, 1 )
2   #define OLED_CMD()    GPIOSetValue( PORT2, 7, 0 )
3   #define OLED_CS_OFF() GPIOSetValue( PORT0, 2, 1 )
4   #define OLED_CS_ON()  GPIOSetValue( PORT0, 2, 0 )
5
6   static void writeData(uint8_t data)
7   {
8       OLED_DATA();
9       OLED_CS_ON();
10      SSPSend( (uint8_t *)&data, 1 );
11      OLED_CS_OFF();
12  }
```

Wysyłanie poleceń i danych do kontrolera wyświetlacza polega na ustawieniu portu GPIO w stan niski dla przesłania polecenia oraz w stan wysoki w celu przesłania danych, a następnie wysłanie 1 bajtu za pomocą interfejsu SPI.

- `OLED_CS_OFF()` - deaktywuje (wyłącza) sygnał CS (Chip Select) dla wyświetlacza OLED (urządzenie nieaktywne)
- `OLED_CS_ON()` - aktywuje (włącza) sygnał CS (Chip Select) dla wyświetlacza OLED (urządzenie aktywne)

5.5.6. Cechy i zalety wyświetlacza OLED w projekcie

- **Wysoka rozdzielczość** – 96x64 px pozwala na prezentację szczegółowych informacji i prostych grafik.
- **Nieskończony kontrast** – technologia OLED zapewnia idealną czerń i wysoką czytelność nawet przy niewielkich rozmiarach.

- **Bogata paleta barw** – do 65536 kolorów (16 bitów na piksel).
- **Szybka komunikacja SPI** – umożliwia dynamiczne odświeżanie ekranu, niezbędne w aplikacjach interaktywnych.
- **Elastyczne zasilanie i łatwy montaż** – szeroki zakres napięć oraz wlutowane goldpiny.

Podsumowanie: Wyświetlacz OLED jest kluczowym elementem interfejsu użytkownika w projekcie – umożliwia prezentację wyników, komunikatów oraz dynamicznych zmian stanu gry w sposób czytelny i atrakcyjny wizualnie.

5.6. Interfejs I2C

I2C (Inter-Integrated Circuit) to synchroniczny, szeregowy protokół komunikacyjny typu master-slave, wykorzystywany do łączenia mikrokontrolerów z układami peryferyjnymi, takimi jak czujniki, pamięci czy ekspandery portów. W projekcie interfejs I2C służy do komunikacji m.in. z czujnikiem oświetlenia ISL29003.

5.6.1. Zasada działania magistrali I2C

Magistrala I2C wykorzystuje dwie linie:

- **SDA** (Serial Data Line) – linia danych,
- **SCL** (Serial Clock Line) – linia zegara.

Obie linie są typu otwarty kolektor, wymagają podciągnięcia do napięcia zasilania przez rezystory (najczęściej 4,7 kΩ), co pozwala na podłączanie wielu urządzeń do wspólnej magistrali.

Transmisja danych odbywa się w postaci komunikatów (messages) podzielonych na ramki (frames):

- **Start Condition** – sygnalizowana przez przejście SDA z wysokiego na niski poziom przy wysokim SCL.
- **Address Frame** – 7- lub 10-bitowy adres slave'a oraz bit R/W (0 – zapis, 1 – odczyt).
- **Data Frames** – każda ramka to 8 bitów danych, przesyłanych z najbardziej znaczącym bitem (MSB) jako pierwszym.
- **ACK/NACK** – po każdej ramce odbiorca potwierdza odbiór przez wystawienie bitu ACK (0) lub NACK (1).
- **Stop Condition** – przejście SDA z niskiego na wysoki poziom przy wysokim SCL.

Adresowanie pozwala na podłączenie do 128 urządzeń slave (przy adresacji 7-bitowej) bez konieczności stosowania dodatkowych linii wyboru, jak w SPI.

5.6.2. Konfiguracja sprzętowa na LPC1343

Aby uruchomić interfejs I2C w trybie master:

```
LPC_SYSCON->PRESETCTRL |= (0x1<<1);          // Reset I2C
LPC_SYSCON->SYSAHBCLKCTRL |= (1<<5);        // Wlaczenie zegara I2C

LPC_IOCON->PIO0_4 &= ~0x3F;
LPC_IOCON->PIO0_4 |= 0x01;                  // SCL
LPC_IOCON->PIO0_5 &= ~0x3F;
LPC_IOCON->PIO0_5 |= 0x01;                  // SDA

LPC_I2C->SCLL = 0x180;                      // Czas trwania stanu niskiego SCL
LPC_I2C->SCLH = 0x180;                      // Czas trwania stanu wysokiego SCL
```

Częstotliwość sygnału zegarowego SCL wyznacza wzór:

$$f_{I2C} = \frac{PCLK}{2 \times (SCLH + SCLL)}$$

Dla PCLK = 72 MHz oraz SCLH = SCLL = 384, uzyskujemy $f_{I2C} \approx 46,9$ kHz.

5.6.3. Rejestry I2C w LPC1343

- **I2CONSET/I2CONCLR** – rejestry ustawień i kasowania bitów kontrolnych.
- **I2STAT** – rejestr statusu (kody stanu transmisji).
- **I2DAT** – rejestr danych (do wysyłania i odbioru bajtów).
- **I2SCLH/I2SCLL** – rejestry ustawiające długość impulsu zegara SCL.
- **I2ADR0-3** – rejestry adresowe dla trybu slave.
- **I2DATA BUFFER** – bufor odbiorczy.

5.6.4. Protokół komunikacji z ISL29003

Czujnik światła ISL29003 komunikuje się przez I2C jako slave (domyślny adres: 0x44). Odczyt danych z czujnika wymaga wysłania adresu rejestru, z którego chcemy czytać (np. adres LSB lub MSB wyniku pomiaru), a następnie odczytu bajtu danych.

Przykładowa funkcja odczytu natężenia światła:

```
uint32_t light_read(void) {
    uint32_t data = 0;
    uint8_t buf[1];
    buf[0] = ADDR_LSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = buf[0];
    buf[0] = ADDR_MSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = (buf[0] << 8) | data;
    return (range * data / width);
}
```

Opis działania:

- Wysłanie adresu rejestru LSB, odczyt młodszego bajtu.

- Wysłanie adresu rejestru MSB, odczyt starszego bajtu.
- Połączenie bajtów do postaci 16-bitowej liczby.
- Przeliczenie wartości na jednostki luksów zgodnie z wybranym zakresem i rozdzielczością czujnika (parametry `range` i `width`).

Podsumowanie: Interfejs I2C zapewnia prostą i niezawodną komunikację z wieloma układami peryferyjnymi na wspólnej magistrali, a dzięki adresowaniu umożliwia obsługę wielu urządzeń bez konieczności dodatkowych linii wyboru. Implementacja obsługi czujnika ISL29003 przez I2C pozwala na szybki i precyzyjny odczyt natężenia światła w czasie rzeczywistym.

5.7. Linijka diod

Linijka diod LED w projekcie jest realizowana za pomocą układu PCA9532, będącego 16-bitowym sterownikiem LED z interfejsem I2C. Umożliwia on indywidualne sterowanie stanem oraz jasnością do 16 diod LED, a także generowanie efektów takich jak miganie czy płynne ściemnianie.

5.7.1. Charakterystyka PCA9532

PCA9532 to specjalizowany ekspander I/O zoptymalizowany do sterowania diodami LED. Najważniejsze cechy:

- 16 wyjść typu open-drain, mogących bezpośrednio sterować diodami LED.
- Możliwość ustawienia każdego wyjścia w jednym z czterech trybów: wyłączony (OFF), włączony (ON), miganie z częstotliwością i współczynnikiem wypełnienia PWM0, miganie z PWM1.
- Dwa niezależnie programowalne generatory PWM (częstotliwość od 0,591 Hz do 152 Hz, 256 poziomów jasności).
- Obsługa I2C do 400 kHz.
- Możliwość pracy kilku układów na jednej magistrali (3 piny adresowe A0-A2).
- Po resecie wszystkie diody są wyłączone (stan wysoki na wyjściach).

5.7.2. Podłączenie i adresacja

Wyjścia LED0–LED15 są bezpośrednio połączone z diodami LED przez rezystory ograniczające prąd. Adres układu na magistrali I2C ustala się przez odpowiednie podłączenie pinów A0–A2 do masy lub zasilania (w projekcie: 0x60).

5.7.3. Sterowanie diodami w programie

W programie do sterowania diodami wykorzystywane są maski bitowe:

- `ledOffMask` – bity ustawione na 1 oznaczają diody wyłączone,
- `ledOnMask` – bity ustawione na 1 oznaczają diody włączone.

W każdej rundzie gry maska diod włączonych jest przesuwana w lewo, aby wizualizować postęp (np. `0x0E`, `0x3C`, `0xF8` itd.).

Najpierw zerowane są wszystkie bity zmiennej `ledStateShadow` przez operację:

```
ledStateShadow &= (ledOffMask & 0xFFFF);
```

Następnie ustawiane są bity odpowiadające diodom do włączenia:

```
ledStateShadow |= ledOnMask;
```

Po ustaleniu stanu wywoływana jest funkcja `setLeds()`, która przekazuje stan do PCA9532 przez I2C.

5.7.4. Przykład ustawiania stanów diod

Tablica `ls[]` przechowuje stany LED w 4 bajtach (po 4 diody na bajt). Funkcja `setLsStates()` ustawia odpowiednie bity:

```
#define IS_LED_SET(bit, x) ( ((x) & (bit)) ? 1 : 0 )
for (int i = 0; i < 4; i++) {
    ls[i] |= (
        (IS_LED_SET(LED4, states) * mode << 0) |
        (IS_LED_SET(LED5, states) * mode << 2) |
        (IS_LED_SET(LED6, states) * mode << 4) |
        (IS_LED_SET(LED7, states) * mode << 6)
    );
    states >>= 4;
}
```

Makro `IS_LED_SET` sprawdza, czy dany bit odpowiadający diodzie jest ustawiony w masce stanu.

Definicje bitów LED

Każda dioda ma przypisany bit:

```
#define LED4  0x0001 // LED0 na plytce
#define LED5  0x0002 // LED1
// ...
#define LED19 0x8000 // LED15
```

Uwagi praktyczne

- PCA9532 posiada wewnętrzny oscylator – nie wymaga dodatkowych elementów zewnętrznych.
- Wyjścia typu open-drain umożliwiają bezpieczne sterowanie diodami LED z różnych napięć (zasilanie układu 2,3–5,5 V).
- Diody LED można ściemniać i migać programowo, co pozwala na efekty wizualne bez obciążania mikrokontrolera.
- Nieużywane wyjścia mogą być wykorzystane jako GPIO.

Podsumowanie: Zastosowanie PCA9532 umożliwia elastyczne i wydajne sterowanie linijką diod LED przez interfejs I2C, z możliwością programowej kontroli jasności, efektów migania oraz łatwej rozbudowy o kolejne linie LED bez zajmowania dodatkowych pinów mikrokontrolera.

5.8. Moduł EEPROM

Do zapisu najwyższego wyniku gry wykorzystujemy pamięć nieulotną typu EEPROM, model 24LC08B, podłączoną do magistrali I2C pod adresami od 0x50 do 0x53. Pamięć ta posiada 1024 bajty podzielone na cztery bloki po 256 bajtów każdy. Operacje zapisu i odczytu realizowane są poprzez interfejs I2C, który należy uprzednio zainicjalizować.

5.8.1. Adresowanie i organizacja pamięci

Adres urządzenia EEPROM 24LC08B składa się z:

- Stałej części adresu (bity b7-b4) ustawionej na 1010,
- Trzech bitów A2, B1, B0 określających blok pamięci i konfigurację pinów adresowych,
- Bitu R/W określającego operację odczytu (1) lub zapisu (0).

Dzięki temu możliwe jest kaskadowanie do czterech bloków pamięci (4 x 256 bajtów), co daje łącznie 1024 bajty.

5.8.2. Funkcje zapisu i odczytu

Do zapisu danych wykorzystywana jest funkcja:

```
1 void eeprom_write(uint8_t* buf, uint16_t offset, uint16_t len);
```

gdzie:

- `buf` – wskaźnik na dane do zapisania,
- `offset` – offset (adres początkowy) względem początku pamięci EEPROM,
- `len` – liczba bajtów do zapisania.

Do odczytu danych służy funkcja:

```
1 void eeprom_read(uint8_t* buf, uint16_t offset, uint16_t len);
```

gdzie:

- `buf` – wskaźnik na bufor, do którego zostaną zapisane odczytane dane,
- `offset` – offset (adres początkowy) względem początku pamięci EEPROM,
- `len` – liczba bajtów do odczytania.

5.8.3. Procedura odczytu

Operacja odczytu wymaga najpierw wysłania adresu wewnętrznego pamięci, z którego chcemy czytać, poprzez zapis na magistrali I2C, a następnie odczytu danych. Przykładowy fragment kodu ilustrujący tę procedurę:

```
1 uint8_t addr = EEPROM_I2C_ADDR1 + (offset / EEPROM_BLOCK_SIZE);
2 uint8_t off = offset % EEPROM_BLOCK_SIZE;
3 I2CWrite((addr << 1), (uint8_t*)&off, 1);
4 for (int i = 0; i < 0x2000; i++); // opoznienie na zakonczenie operacji
5 I2CRead((addr << 1), buf, len);
```

- `addr` – adres urządzenia EEPROM na magistrali I2C, zależny od bloku pamięci,
- `off` – offset wewnątrz bloku pamięci,
- `I2CWrite` – funkcja wysyłająca adres odczytu,
- `I2CRead` – funkcja odczytująca dane z pamięci EEPROM.

5.8.4. Opis działania

- Po wysłaniu adresu wewnętrznego następuje opóźnienie (np. pętla `for`), które pozwala na zakończenie wewnętrznego cyklu zapisu lub przygotowanie danych do odczytu.
- Następnie następuje odczyt żądanej liczby bajtów z pamięci EEPROM.
- Podczas zapisu EEPROM 24LC08B automatycznie wykonuje wewnętrzny cykl zapisu, który trwa do kilku milisekund. W tym czasie urządzenie nie odpowiada na kolejne żądania na magistrali I2C.
- Możliwe jest wykonywanie operacji page write (zapis do 16 bajtów na raz), gdzie adres wewnętrzny automatycznie się inkrementuje, a po przekroczeniu rozmiaru strony następuje nadpisanie danych od początku strony.

Zalecenia i uwagi

- Pin WP (Write Protect) powinien być podłączony do masy (VSS), aby umożliwić zapis do całej pamięci.
- Należy uwzględnić czas wewnętrznego cyklu zapisu EEPROM (maksymalnie 5 ms) i nie inicjować kolejnych operacji zapisu przed jego zakończeniem.
- W przypadku odczytu sekwencyjnego można czytać kolejne bajty bez ponownego wysyłania adresu, aż do końca pamięci, po czym adresowanie „zawija się" do początku.
- Podczas programowania strony (page write) należy uważać, aby nie przekroczyć 16 bajtów, gdyż nadmiar danych spowoduje nadpisanie początkowych bajtów strony.

Podsumowanie

Moduł EEPROM 24LC08B zapewnia trwałe przechowywanie danych, takich jak rekordowe wyniki gry, dzięki czemu możliwe jest ich zachowanie po wyłączeniu zasilania. Komunikacja przez interfejs I2C oraz mechanizmy adresowania blokowego umożliwiają efektywne zarządzanie pamięcią i prostą integrację z mikrokontrolerem LPC1343.

5.9. Akcelerometr MMA7455L

Akcelerometr MMA7455L firmy Freescale Semiconductor to zaawansowany trójosiowy cyfrowy czujnik MEMS (Micro-Electro-Mechanical Systems), umożliwiający precyzyjny pomiar przyspieszenia wzdłuż osi X, Y i Z. W projekcie wykorzystano komunikację przez interfejs I2C. Urządzenie znajduje szerokie zastosowanie w elektronice użytkowej, systemach nawigacyjnych, aplikacjach medycznych oraz przemysłowych do detekcji ruchu, orientacji, wstrząsów, swobodnego opadania i analizy wibracji.

5.9.1. Główne cechy MMA7455L

- Cyfrowe wyjście (I2C lub SPI, w projekcie użyto I2C)
- Pomiar przyspieszenia w trzech osiach (X, Y, Z)
- Zakresy pomiarowe: $\pm 2\,g$, $\pm 4\,g$, $\pm 8\,g$ (w projekcie: $\pm 2\,g$)
- Czułość: 64 LSB/g przy 2g (tryb 10-bitowy)
- Możliwość generowania dwóch niezależnych przerwań (wykrywanie wstrząsów, swobodnego opadania, wibracji)

Interfejsy komunikacyjne

- **I2C:** Standard mode (100 kHz), Fast mode (400 kHz)
- **SPI:** Do 8 MHz
- **Przerwania:** Dwa niezależne wyjścia (INT1, INT2)
- **Funkcje przerwań:** Level detection, pulse detection, freefall detection

Adresacja I2C

- **Adres 8-bitowy:** `0x3A` (zapis), `0x3B` (odczyt)
- **Częstotliwość zegara:** do 400 kHz (Fast mode I2C)

Rejestry danych przyspieszenia

Nazwa rejestru	Adres hex	Typ	Opis
XOUT10	0x00	R	X-axis acceleration data (10-bit, LSB)
XOUT8	0x06	R	X-axis acceleration data (8-bit)
YOUT10	0x02	R	Y-axis acceleration data (10-bit, LSB)
YOUT8	0x07	R	Y-axis acceleration data (8-bit)
ZOUT10	0x04	R	Z-axis acceleration data (10-bit, LSB)
ZOUT8	0x08	R	Z-axis acceleration data (8-bit)

Tabela 5.2: Rejestry danych przyspieszenia

Funkcja odczytu danych przyspieszenia

```
1  /**
2   * @brief Odczyt przyspieszenia z wszystkich osi (8-bit)
3   * @param x Wskaźnik na wartość osi X
4   * @param y Wskaźnik na wartość osi Y
5   * @param z Wskaźnik na wartość osi Z
6   */
7  void acc_read(int8_t *x, int8_t *y, int8_t *z)
8  {
9      uint8_t buf[1];
10
11     // Oczekiwanie na gotowość nowych danych
```

```
12      while ((getStatus() & ACC_STATUS_DRDY) == 0);
13
14      // Odczyt osi X
15      buf[0] = ACC_ADDR_XOUT8;
16      I2CWrite(ACC_I2C_ADDR, buf, 1);
17      I2CRead(ACC_I2C_ADDR, buf, 1);
18      *x = (int8_t)buf[0];
19
20      // Odczyt osi Y
21      buf[0] = ACC_ADDR_YOUT8;
22      I2CWrite(ACC_I2C_ADDR, buf, 1);
23      I2CRead(ACC_I2C_ADDR, buf, 1);
24      *y = (int8_t)buf[0];
25
26      // Odczyt osi Z
27      buf[0] = ACC_ADDR_ZOUT8;
28      I2CWrite(ACC_I2C_ADDR, buf, 1);
29      I2CRead(ACC_I2C_ADDR, buf, 1);
30      *z = (int8_t)buf[0];
31  }
```

Opis działania:

- Inicjalizowany jest jednobajtowy bufor do komunikacji I2C.
- Program oczekuje na ustawienie flagi gotowości danych (`ACC_STATUS_DRDY`) przez funkcję `getStatus()`.
- Następnie dla każdej osi (X, Y, Z):
  - Do bufora wpisywany jest adres rejestru danej osi (`ACC_ADDR_XOUT8`, `YOUT8`, `ZOUT8`).
  - Wysyłane jest żądanie odczytu przez I2C.
  - Odczytana wartość konwertowana jest na `int8_t` i zapisywana pod wskazany adres.

Inicjalizacja akcelerometru

Funkcja `acc_init()` ustawia tryb pracy na pomiarowy i wybiera zakres pomiarowy:

```
1  /**
2   * @brief Inicjalizacja akcelerometru MMA7455L
3   */
4  void acc_init (void)
5  {
6      setModeControl( (ACC_MCTL_MODE (ACC_MODE_MEASURE)
7          | ACC_MCTL_GLVL(ACC_RANGE_2G) ));
8  }
```

- `ACC_MCTL_MODE(ACC_MODE_MEASURE)` – ustawia tryb pracy na pomiarowy (measurement mode).
- `ACC_MCTL_GLVL(ACC_RANGE_2G)` – ustawia zakres pomiarowy na $\pm 2g$, co jest typowe dla aplikacji użytkowych.
- Parametry są przekazywane do rejestru kontrolnego przez funkcję `setModeControl()`.

Przetwarzanie i interpretacja danych

Odczytane wartości są liczbami ze znakiem (`int8_t`), których zakres odpowiada wybranemu zakresowi pomiarowemu. Przykładowo, dla zakresu $\pm 2g$, wartość 64 odpowiada przyspieszeniu 1g (czułość 64 LSB/g). Dzięki temu możliwe jest wykrywanie ruchów, przechyłów, wstrząsów czy swobodnego opadania układu.

Uwagi praktyczne

- W projekcie akcelerometr służy m.in. do wykrywania przechylenia płytki (np. do resetowania rekordu).
- Dwa wyjścia przerwań mogą być wykorzystane do szybkiego reagowania na zdarzenia bez konieczności ciągłego odpytywania czujnika.
- Dzięki szerokiemu zakresowi napięć oraz wbudowanym układom poziomującym, moduł jest kompatybilny z większością mikrokontrolerów 3,3 V i 5 V.

Podsumowanie: MMA7455L umożliwia precyzyjny pomiar przyspieszenia w trzech osiach, jest łatwy w integracji dzięki interfejsowi I2C i stanowi uniwersalny komponent do detekcji ruchu, orientacji oraz zdarzeń dynamicznych w systemach wbudowanych.

5.10. Czujnik oświetlenia ISL29003

ISL29003 to zintegrowany czujnik światła wyposażony w 16-bitowy przetwornik ADC oraz programowalny zakres pomiarowy w luksach. Czujnik obsługuje sprzętowe wyjście przerwania, które pozostaje aktywne w stanie niskim do momentu zresetowania go przez hosta za pośrednictwem interfejsu I2C. ISL29003 został zaprojektowany do pracy z napięciem zasilania od 2,5 V do 3,3 V. Jest stale podłączony do magistrali I2C, a jego wyjście przerwania może być podłączone do pinu PIO2_5 mikrokontrolera.

5.10.1. Odczyt natężenia światła – funkcja `light_read()`

Do odczytu natężenia światła wykorzystywana jest funkcja `light_read()`, która komunikuje się z czujnikiem poprzez protokół I2C:

```
uint32_t light_read(void)
{
    uint32_t data = 0;
    uint8_t buf[1];
    buf[0] = ADDR_LSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = buf[0];
    buf[0] = ADDR_MSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = (buf[0] << 8 | data);
    return (range * data / width);
}
```

Opis działania funkcji:

- **Odczyt młodszego bajtu (LSB):**
  - `buf[0] = ADDR_LSB_SENSOR;` – ustawia bufor na adres młodszego bajtu danych czujnika.
  - `I2CWrite(LIGHT_I2C_ADDR, buf, 1);` – wysyła adres rejestru do czujnika przez I2C.
  - `I2CRead(LIGHT_I2C_ADDR, buf, 1);` – odczytuje młodszy bajt danych z czujnika i zapisuje go w `buf[0]`.
  - `data = buf[0];` – zapisuje odczytane dane w zmiennej `data`.

- **Odczyt starszego bajtu (MSB):**
  - `buf[0] = ADDR_MSB_SENSOR;` – ustawia bufor na adres starszego bajtu danych.
  - `I2CWrite(LIGHT_I2C_ADDR, buf, 1);` oraz `I2CRead(LIGHT_I2C_ADDR, buf, 1);` – wysyłają i odczytują starszy bajt danych.
  - `data = (buf[0] « 8 | data);` – łączy młodszy i starszy bajt w jedną 16-bitową liczbę.
- **Przeliczenie wartości:**
  `return (range * data / width);` – przelicza wartość z czujnika na luksach, używając zakresu `range` i rozdzielczości `width`.

5.10.2. Włączanie i konfiguracja czujnika – funkcja `light_enable()`

Do aktywacji i konfiguracji czujnika służy funkcja:

```
void light_enable (void)
{
    uint8_t buf[2];
    buf[0] = ADDR_CMD;
    buf[1] = CMD_ENABLE;
    I2CWrite(LIGHT_I2C_ADDR, buf, 2);
    range = RANGE_K1;
    width = WIDTH_16_VAL;
}
```

Opis działania funkcji:

- `buf[0] = ADDR_CMD;` – stała o wartości 0x00, definiująca adres rejestru komendy w czujniku światła.
- `buf[1] = CMD_ENABLE;` – stała o wartości (1 « 7), reprezentująca komendę włączenia urządzenia.
- `I2CWrite(LIGHT_I2C_ADDR, buf, 2);` – wysyła przez I2C dwubajtowy pakiet (adres rejestru i komenda) na adres czujnika. Przyjmuje trzy argumenty: adres urządzenia I2C (LIGHT_I2C_ADDR), wskaźnik do bufora danych (buf), oraz liczbę bajtów do wysłania
- `LIGHT_I2C_ADDR` – (0x44 « 1) – adres I2C urządzenia, do którego chcemy wysłać dane
- `range = RANGE_K1;` – ustawia zakres pracy urządzenia, co definiuje czułość lub zakres pomiaru czujnika światła.
- `width = WIDTH_16_VAL;` – ustawia szerokość (np. 1 « 16), co oznacza 16-bitową rozdzielczość pomiaru.

Uwagi praktyczne i integracja sprzętowa

- ISL29003 jest stale podłączony do magistrali I2C i komunikuje się poprzez adres `LIGHT_I2C_ADDR` (0x44 « 1).
- Wyjście przerwania czujnika może być podłączone do pinu PIO2_5 mikrokontrolera. Przerwanie pozostaje aktywne (stan niski), dopóki nie zostanie zresetowane przez hosta za pomocą I2C.
- Czujnik umożliwia szybkie i precyzyjne pomiary natężenia światła, co jest wykorzystywane m.in. do adaptacyjnego sterowania jasnością wyświetlacza lub jako źródło ziarna dla generatora losowego w aplikacji.

6. Analiza skutków awarii (FMEA)

Tabela 3. Analiza FMEA (Failure Mode and Effect Analysis).

Ryzyko	Prawdopodobieństwo	Znaczenie	(Samo)Wykrywalność	Reakcja	Iloczyn
Awaria timera	0.1	krytyczne (10)	Brak	Brak	1.0
Awaria dźwięku	0.1	niegroźne (2)	Brak	Brak	0.2
Awaria joysticka	0.2	krytyczne (10)	Brak	Brak	3.0
Awaria wyświetlacza	0.2	krytyczne (10)	Brak	Brak	2.0
Awaria linijki diod	0.1	niegroźne (2)	Brak	Brak	0.1
Awaria EEPROM	0.1	średnie (5)	Brak	Brak	1.0
Awaria akcelerometru	0.3	niegroźne (2)	Brak	Brak	0.5
Awaria czujnika oświetlenia	0.3	niegroźne (2)	Brak	Brak	0.1